Using Symbolic Execution to Discretize State Spaces for Reinforcement Learning

Mohsen Ghaffari[1], Mahsa Varshosaz[1],
Einar Broch Johnsen[2], and Andrzej Wąsowski[1]

[1] IT University of Copenhagen, Copenhagen, Denmark,
[2] University of Oslo, Oslo, Norway

Abstract

One of the most significant challenges in Reinforcement Learning lies in its slow convergence rate when dealing with continuous state spaces or sparse rewards. Discretizing the state spaces can add generalization to learning and enable exploiting experiences gained earlier which may improve this issues, but it needs a discretization which is fine enough to capture critical information about the state, but does not lead to a combinatorial explosion. In pursuit of this goal, we explore the possibility of extracting representations from the agent's dynamics by symbolically executing models of the environment and the agent.

1 Introduction

Reinforcement learning trains policies to achieve goals through interactions with the environment. Discrete reinforcement learning algorithms store learned behaviors in tabular representations, which is not suitable for large or continuous state spaces often encountered in reality. Continuous state spaces typically require discretization to apply these methods effectively. Alternatives like Deep Reinforcement Learning are available for managing continuous spaces. But, they suffer from issues related to computational complexity, sparse rewards, noisy environments, explainability, and no convergence guarantees.

Discretization of the continuous state space is a common and useful approach as it can generalize and reduce the state space. Various techniques for dicretization exists; basis functions [1], tile coding [3], and vector quantization [4]. While each of these offers benefits, they ignore non-linear dependencies in the state space even though non-linear behaviors are common in control systems, which leads to fine-grained partitions to yield policies that deviate significantly from the encompassing partition, among others.

Discretization process partitions the continuous state space into finite subsets, each represents a discrete state of the agent. It is crucial that each of these subsets captures meaningful information about the environment. The choice of how to perform discretization depends on the specific problem at hand. It is worth noting that, even after discretization, the reinforcement learning process typically involves repeated interactions with the environment over a limited timeframe. For this reason, agent may not learn a proper policy for entire state space of the environment. This can pose challenges, particularly in critical systems where high reliability is essential. Therefore, ensuring that even the smallest partitions have distinct states is vital, as it allows us to guarantee that the agent will learn an appropriate policy for each unique state.

In this work, we investigate extracting the representation from the agent's dynamics using symbolic execution of the environment models. *Symbolic execution* is a program analysis technique that allows for exploring all possible paths of execution of a program by using symbolic values instead of concrete values as inputs [2]. In symbolic execution, a program is executed symbolically by substituting its input variables with symbolic expressions representing all possible values they could take. As the program executes, the symbolic expressions are manipulated

to generate a set of path constraints, called *Path Condition*, which represent the conditions that must hold for each path of execution to be taken. Path conditions are built by accumulating the branch conditions encountered during the symbolic execution of the program. The hypothesis of this work is that partitioning obtained by path conditions of a single step through the environment model provides a useful discretization for reinforcement learning. We are using symbolic execution to extract all of the states that follow the same syntactical path in the simulator of the environment for all actions. Note that, in this work we assume the action set is discrete.

2 Symbolic Execution of Reinforcement Learning Problems

In this work, we assume single agent problems and the environment is specified as a computer program. The program is implementing a single step transition in the environment and corresponding reward. We can formulate it as $Env : \mathcal{T} \wedge \mathcal{R}$, where $\mathcal{T} \in \overline{\mathcal{S}} \times \mathcal{A} \to \text{pdf}\ \overline{\mathcal{S}}$ is the transition probability function, $\overline{\mathcal{S}}$ is a possibly uncountable set of states, $\mathcal{A}$ is a finite set of actions, and $\mathcal{R} \in \overline{\mathcal{S}} \times \mathcal{A} \times \overline{\mathcal{S}} \to \mathbb{R}$ is the reward function. In the tabular algorithms, it is not common to record $\overline{\mathcal{S}}$ in the table. Instead we map it into an observable state $\mathcal{O} \in \overline{\mathcal{S}} \to \mathcal{S}$ and record the observable state in the table. We employ symbolic execution to analyse the environment, then discretize the state space using the analysis results. The obtained partitioning allows us to define $\mathcal{O}$ properly and the entire process can be automated and generic.

When a program runs, it follows specific paths dictated by conditions like if-statements or loops. Symbolic execution operates on the program with symbolic variables instead of concrete values, which allows to explore multiple possible scenarios simultaneously. Each of these scenarios corresponds to a specific path condition. As the program symbolically unfolds, it generates symbolic formulas that represent relationships between symbols and program conditions, essentially expressing the program's behavior in a mathematical form. Therefore, for a program P: $I \to O$, where $I = \{v_1, v_2, \ldots, v_k\}$ is the set of input variables and O is the set of output variables of the program, path condition is a logical expression $\phi(I')$ that satisfies only one branch of the program and $I' = \{sym_{v_1}, sym_{v_2}, \ldots, sym_{v_k}\}$, where each sym_{v_i} is the symbolic variable corresponding to the i-th variable in the I. In presence of loops and recursion symbolic execution does not terminate. To halt symbolic execution, we can set a predefined timeout or iteration limit, forcing the analysis to stop after a specified number of iterations. Although this technique rises the approximation of the obtained path conditions, we still would have a appropriate match between the dynamics of the program and expressions in each path condition.

Ideally two continuous states $\overline{s_1},\ \overline{s_2} \in \overline{\mathcal{S}}$ should be in the same partition (have the similar behavior with the agent) iff

$$\forall a \in \pi^*.\ \mathcal{T}(\overline{s_1}, a) = \mathcal{T}(\overline{s_2}, a) \wedge \mathcal{R}(\overline{s_1}, a) = \mathcal{R}(\overline{s_2}, a). \tag{1}$$

where π^* is the optimal policy. Then the same action is optimal for each state in the partition.

According to Eq (1), we must find concrete states with similar behavior for a given action to group them into a partition. As this requirement is very strong, we will use a relaxation instead. We consider states to be equivalent if they trigger the same execution path in the system dynamic as implemented in a simulator. This means they have the same value for the predicates on the program execution path. To this end, we use symbolic execution to extract all possible path conditions of the Env, when $\overline{\mathcal{S}}$ is a symbolic input and $\mathcal{A}$ is a concrete value. This results a set of path conditions for each action. In our specification, each path condition will be a logical expression $\phi(\overline{\mathcal{S}})$. This means, for any concrete value in $\overline{\mathcal{S}}$ that satisfies ϕ and the corresponding action, there is one and only one path in Env. Since these path conditions correspond to an action in $\mathcal{A}$, and Eq (1) quantifies over all actions in $\mathcal{A}$, we do cross-set satisfiability checking for all the sets.

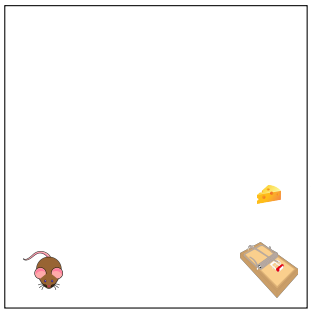

Figure 1: The environment of problem

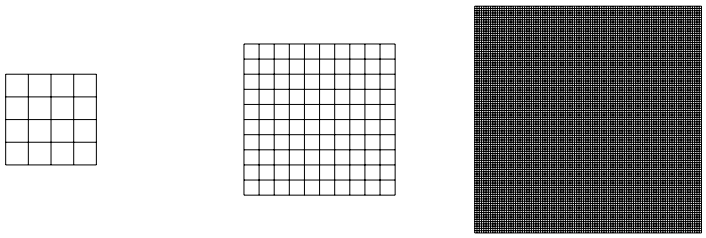

Figure 2: Discretization using a tiling approach for a room sizes of 4×4, 10×10, and 100×100 from left to right.

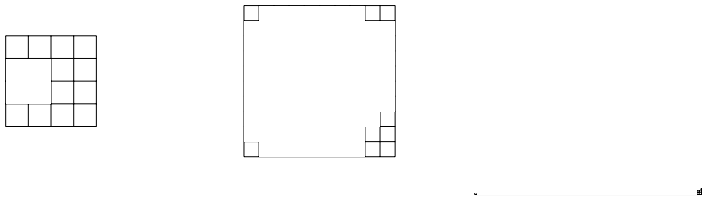

Figure 3: Discretization using a our approach for a room sizes of 4×4, 10×10, and 100×100 from left to right.

For example, consider a room with dimensions $m \times n$, where a mouse is in the bottom-left corner, a mousetrap is in the bottom-right corner, and a piece of cheese is just above the mousetrap (Fig. 1). The mouse can move in four primary directions: U, D, R, L. The goal is for the mouse to reach the cheese while avoiding the mousetrap. We can define the state as a tuple representing the mouse's position in the room. Discretizing using a tiling approach creates a grid of size $m \times n$ (Fig. 2), we have a total of $m \times n$ observable states. For this example, our approach generates a maximum of 15 states (Fig. 3). This example provides an insight to the outcomes of the proposed approach, and the evaluation is in progress.

3 Acknowledgments

Partially funded by DIREC (Digital Research Centre Denmark), a collaboration between the eight Danish universities and the Alexandra Institute supported by the Innovation Fund Denmark.

References

[1] Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6, 2005.

[2] James C King. Symbolic execution and program testing. *Communications of the ACM*, 19(7):385–394, 1976.

[3] Stephen Lin and Robert Wright. Evolutionary tile coding: An automated state abstraction algorithm for reinforcement learning. In *Proceedings of the 8th AAAI Conference on Abstraction, Reformulation, and Approximation*, pages 42–47, 2010.

[4] Christos N Mavridis and John S Baras. Vector quantization for adaptive state aggregation in reinforcement learning. In *2021 American Control Conference (ACC)*, pages 2187–2192. IEEE, 2021.