Synthesizing Understandable Strategies

Peter Backeman[1][0000−0001−7965−248X]

Mälardalen University, Västerås, Sweden
peter.backeman@mdu.se

Abstract. The result of reinforcement learning is often obtained in the form of a q-table mapping actions to future rewards. We propose to use SMT solvers and *strategy trees* to generate a representation of a learned strategy in a format which is understandable for a human. We present the methodology and demonstrate it on a small game.

Keywords: Synthesizing Strategies · Reinforcement Learning · SMT

1 Introduction

Reinforcement learning has in the last decade gained enormous popularity for creating an AI agent learning an optimal strategy. The result is often obtained in the form of a q-table, mapping, for each state, every action to an expected future reward. While this is useful for a computer to execute it, for a human it is inconvenient. We propose to use SMT solvers to generate a representation of a learned strategy in a format which is *understandable*, by limiting the representation to a pre-defined format. Moreover, we introduce *strategy trees*, to enable a step-wise refinement process. Generating strategies is a well-researched problem, but In contrast to other recent work, e.g., [4], we emphasize trying to generate an understandable representation of a strategy. We demonstrate an approach on a simple example which we hope to extend to more challenging problems.

Running Example: Nim is a classic game with many variants. In this abstract we focus on a specific version for simplicity, where the game starts with 21 sticks being placed in a row. Each player takes turn removing *one or two sticks*, and the winning player is the one who removes the final stick.

2 Reinforcement Learning

Reinforcement learning is an approach where an optimal strategy is learned by interaction with an environment [3]. The programmer only provides a definition of the state-space, the possible actions and rewards for reaching certain states[1]. For our example, we consider a state space of a single variable $s \in [1, 21]$ indicating the number of remaining sticks. In each state there are two possible actions

[1] As well as a set of meta-parameters to the learning algorithm, e.g., learning rate.

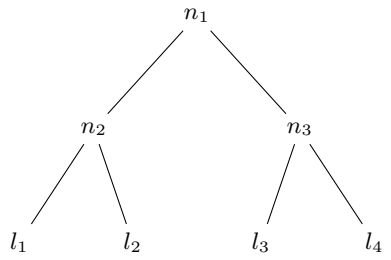

Fig. 1. AST for function template with three internal nodes $\{n_1, n_2, n_3\}$ and five leaf nodes $\{l_1, \ldots, l_5\}$

```
(assert (=> (= n2 0) (and
    (= n2val_0 (+ l1val_0 l2val_0))
    (= n2val_1 (+ l1val_1 l2val_1))
    (= n2val_2 (+ l1val_2 l2val_2))
    (= n2val_3 (+ l1val_3 l2val_3))
    (= n2val_4 (+ l1val_4 l2val_4))
)))
```

Fig. 2. Excerpt of SMT formula.

a_1, a_2, picking one or two sticks [2]. A reward of one is given when winning (i.e, reaching $s = 0$), and minus one when losing (i.e., opponent reaches $s = 0$).

By running a straightforward reinforcement learning algorithm, we can obtain a q-table as shown in Table 1, giving an optimal strategy for playing Nim. For example, it shows that when there are seven sticks left ($s = 7$), taking one stick ($a_1 = 0.90$) is preferable to taking two sticks ($a_2 = -0.90$).

Table 1. Q-table of Nim player.

s	1	2	3	4	5	6	7	8	9	10	11
a_1	1.00	-1.00	-1.00	0.95	-0.91	-0.95	0.90	-0.76	-0.90	0.86	-0.78
a_2	n/a	1.00	-1.00	-0.98	0.95	-0.95	-0.90	0.90	-0.90	-0.74	0.86
s	12	13	14	15	16	17	18	19	20	21	
a_1	-0.86	0.81	-0.76	-0.80	0.77	-0.70	-0.67	0.74	-0.47	-0.72	
a_2	-0.86	-0.75	0.81	-0.81	-0.62	0.77	-0.76	-0.70	0.74	-0.43	

3 SMT Synthesis

Satisfiability Modulo Theories (SMT) is a technique of finding models of formulas defined over a Boolean structure combined with theory literals [1]. In this section we present a method for searching functions, constrained by a template, to find a function which corresponds to a function relating inputs to outputs. A template is shown in Fig. 1. It is an AST with . We restrict each internal node to be one of the operators $+, -, *, \%$ [3], and each leaf to be set either the input value (x) or a constant ($c \in [-10, 10]$). The restrictions on the AST should be set in such a way to balance expressability and understandability.

[2] For simplicity, action a_2 is forbidden when $s = 1$, i.e., it is impossible to pick more sticks than remaining.

[3] Where % is the remainder operator.

Given an AST template and input/output-pairs, we can formulate an SMT query which yields a model with assignments to leaves and internal nodes such that the function computes as desired. We sketch the formulation here:

- Each internal node has an integer variable defining which operator it is,
- Each leaf has two variables: one indicating whether it is the input value or a constant; the second the value of the (potential) constant,
- For each input/output-pair, each node and leaf is given a integer variable which should equal the value of the node or leaf given the specific input.

For example, encoding that if node n_1 is addition (corresponding integer value is zero), then the value of n_2 should be equal to the sum of the values of l_1 and l_2 (with five input/output-pairs) is shown in Fig. 2.

We can encode the q-table in Table 1 (with symmetry-breaking constraints), and solve it using the SMT-solver Z3 [2]. After a long time (hours), it returns a model $\{n_1 = +, n_2 = +, n_3 = \%, l_1 = -1, l_2 = 0, l_3 = x, l_4 = 3\}$ which corresponds to the function $(-1 + 0) + (x\%3) = (x \ \% \ 3) - 1$.[4] This function condenses the strategy into a understandable format.

4 Strategy Trees

As finding the function takes a long time, we introduce an alternative approach: *strategy trees*. A tree over a set of input/output-pairs consists of a root r node with a set of children C, s.t. every edge from r to $c \in C$ is labelled with a function over the input variable, and every leaf is labelled with an output.

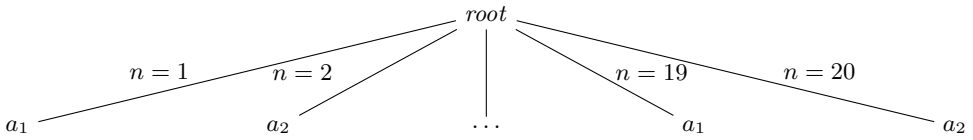

Fig. 3. (Partial) naive strategy tree for Nim game.

Intuitively, each child corresponds to a sub-strategy: its edge-label states *when* it should be applied, and the node-label states *what* action should be made. A naive strategy tree based on the q-table in our running example is obtained by creating one node for each row in the table, see Fig. 3. However, since such a tree is note very useful, we introduce a *merge*-operator, transforming a strategy tree into an equivalent one. We can merge two children if they have the same node label and we can find a edge label which identifies both sub-strategies and no other strategy. We can find functions using the approach presented in Sec. 3.

[4] The subtraction of one comes from the action space being defined as $\{a_1 = 0, a_2 = 1\}$ instead of the number of sticks removed ($\{a_1 = 1, a_2 = 2\}$).

For example, two children with identical node-labels and edge-labels $n = 1$ and $n = 2$ could be merged to a child with edge-label $1 \leq n \wedge n \leq 2$.[5]

We can apply the merge-operator repeatedly to obtain a reduced strategy tree. If we perform this strategy on the tree in Fig. 3, after a few seconds of merging, we obtain the tree shown in Fig. 4, a more succinct representation of the same information. Of course, in which order the operator is applied is important, currently a naive approach is used (enumerate all pairs of children with same node label, and try to merge in order).

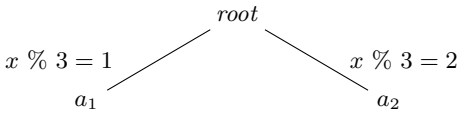

Fig. 4. Reduced strategy tree for Nim game.

5 Conclusions

We presented strategy trees and show we can use them to step-wise synthesize an understandable strategy representation. In future work, we want to flesh out the theory, e.g., considering more input variables and trees with greater depth. Moreover, we will look into different templates for the edge labels. We also wish to explore more use cases and compare different merging strategies and study the scaling of the approach. It is also interesting to introduce a splitting operator to allow the search to go in two directions (where a single split might allow for many merges). **Acknowledgements**: This work was supported by the Knowledge Foundation in Sweden through the ACICS project (20190038).

References

[1] Clark Barrett and Cesare Tinelli. "Satisfiability Modulo Theories". In: *Handbook of Model Checking*. Ed. by Edmund M. Clarke et al. Cham: Springer International Publishing, 2018, pp. 305–343. ISBN: 978-3-319-10575-8.

[2] Leonardo De Moura and Nikolaj Bjørner. "Z3: An efficient SMT solver". In: *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2008, pp. 337–340.

[3] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018.

[4] Kaisheng Wu et al. "Automatic Synthesis of Generalized Winning Strategies of Impartial Combinatorial Games Using SMT Solvers". In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*. Ed. by Christian Bessiere. International Joint Conferences on Artificial Intelligence Organization, 2020, pp. 1703–1711.

[5] Interval constraints are added on edges, limiting the functions domains for efficiency.