Using Reinforcement Learning to Automate Heuristics Searching

Haobei Song

November 2018

Abstract

Model checking is in general an undecidable problem, which however plays vital roles in computer hardware related and safety critical software industries. Difficult industrial problems are solved in an ad hoc approach by combining multiple verification tools and trial-and-failure tuning. This approach is largely driven by hard coded heuristics in the verification tools and often inefficient in terms of time and resources. In addition, the same heuristics might not apply to another problem instae even when they are similar. This paper will start looking at model checking problem from the the top level view as an undecidable problem and reason about how heuristics work in industry from a formal mathematical point of view. A study on the improvement of PDR(IC3) [1, 5] algorithm by letting computer learn the heuristics using reinforcement learning technique is shown in this paper along with its evaluation on computer generated problems. The idea of using reinforcement learning to learn heuristics for a specific domain of problems has the potential to change the paradigm of tackling industrial hardware and software verification problems.

1 Introduction

Model checking inevitably addresses NP-hard problems with constraints or even undecidable problems. However, model checking in industry has developed many practical techniques to facilitate its application in industry. These techniques often consists of large amount of heuristics to address a certain subset of the model checking problems. The heuristics are usually found in an ad hoc, trial and failure approach, therefore it is difficult to formalize the methodology and extend to a different set of problem.

My proposal is to draw the idea from the rapid development of reinforcement learning research, and use learning algorithms to learn the state-of-art heuristics for a set of model checking problems drown from a specific industrial area. This approach generalize to deal with different set of problems as new heuristics can be learned by learning a different reinforcement learning model[4, 6].

This idea of using reinforcement learning technique to find better heuristics can be applied to SAT solvers, SMT solvers, Interpolation based model checking

and IC3 or property directed reach-ability algorithms. In the IC3 algorithm proposed by A. Bradley [1, 2] and further studied by the name of Property Directed Reachability [5], A key step is to find the minimum inductive sub-clauses, which if improved could speed up the model checking by a large amount. Since finding minimum inductive sub-clauses generally requires exponential time, A. Bradley made a compromise to find a "small" inductive sub-clauses(stop after three failure of dropping literals). This solution can be considered as a expedient measure and is a bottleneck for the IC3 algorithms. If one can sufficiently minimize the size of the inductive sub-clauses in a reasonable amount of time, IC3 could improve tremendously as the size of the clauses in the learned formulae sequence becomes small and all the other operations in the algorithm could run faster.

2 What makes heuristics magic

Though model checking is undecidable, in practice the problem space people strive to solve is heavily constrained. However this will not reduce the complexity of the problems as the constraints are too involved to be captured by formal mathematical descriptions. For example, problem sets generated from a specific chip manufacturer. The problems are constrained in terms of the limited types of computations and hardware structures being used required by such chip but to formulate such constraints in rigorous logic formulae is beyond our reach. This leaves solving such constraints impossible in practice as one can not even formulate such problems. Heuristics come into play as a set of "solutions" to these problems as with a number of sophisticated heuristics, the problems can be reduced to almost decidable or polynomial class. One example is the DDPL algorithm which prevails in SAT solving community. SAT problems is NP-complete but some SAT solve can solve certain set of industrial problems very efficiently. And the reason behind of this is a specific set of industrial problems will demonstrate some structural properties such as the locality of propositional variables or the repeated pattern in the CNF of the SAT problem instances.

The problem with using heuristics is to find them for a set of problems generated from a specific domain. Assume there exists such set of heuristics of size L which is much smaller of the size of the problem, the problem space for finding the right set of heuristics is still 2^L by exhaustive search which is what almost all the people are doing. Not much analysis can be done with reasoning about what heuristics to select and which should be abandoned as the underlying logic is broken and heuristics is built on top of such incomplete logic. What is even worse is that the same heuristics often do not work for another set of problems even though they are similar. This problem lies in the fact that no formal analysis logic is available and therefore no generalization can be made.

In this paper an "approximate" logic is proposed. At first look, "approximate" logic is not logic anymore but this can be remedied by using machine learning

and let the computer do the case-by-case reasoning about such "logic". The idea is to use a meta machine learning algorithm to train a reinforcement learning agent to learn about finding heuristics more efficiently. Such agent can be transferred to another set of similar problems as well once the learning agent converges and is able to reason about the structure of the domain specific set of problems presented to it.

3 Reinforcement Learning

Reinforcement Learning is a class of learning problems in which a reinforcement learning agent receives (partially) the dynamic change from the stochastic environment and takes actions which interact with the environment. The goal of the learning agent is to find the optimal strategy to take actions based on the observations received [7]. The environment is often modeled as a (partial) Markov Decision Process (PMDP or MDP). There are generally two types of reinforcement learning algorithm in terms of how the agent perceive the environment, namely model based and model free reinforcement learning.In model based RL setting, the agent learns the model of the environment and make actions based on the learned transition model of the environment. In contrast, the model free algorithm have no such mechanism in its learning process. There are also algorithms built as a mixture of the two. [6]

3.1 Deep Neural Networks

3.1.1 Recurrent Neural Network

Note The RNN part is from my previous project The structure of a typical recurrent neural network is as follows where each neuron or unit has a hidden state and the output of the previous state is fed as the input of the current state.

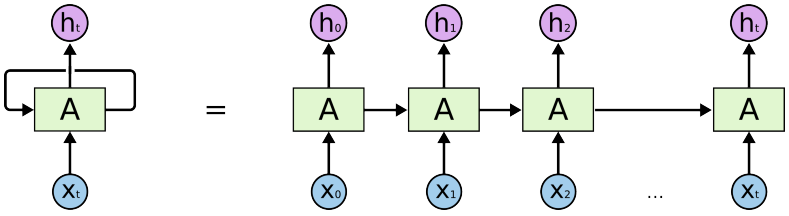
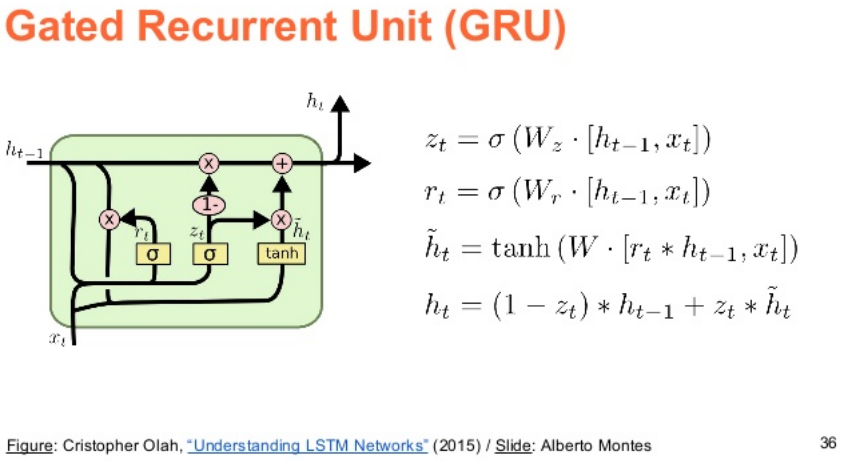


Figure 1: Unrolled Recurrent Unit http://colah.github.io/posts/2015-08-Understanding-LSTMs/img/RNN-unrolled.png

There are a variety of recurrent structures in neural network literature and LSTM(Long-Short-Term-Memory) and GRU(Gated Recurrent Unit) are the two widely used recurrent units. According to the time-invariant assumption

of the Markov decision process, long term memory should not be addressed as the future state only depends on the previous state. When such time-invariant assumption is relaxed, long term memory can help the model learn from the experience as similar concept has been widely used to facilitate convergence of a RL model by caching the state-action-reward-state-action tuples and sampling from past "experience" to update RL model.



The design principle of a GRU unit can be found in [3]

3.1.2 Convolutional Neural Network

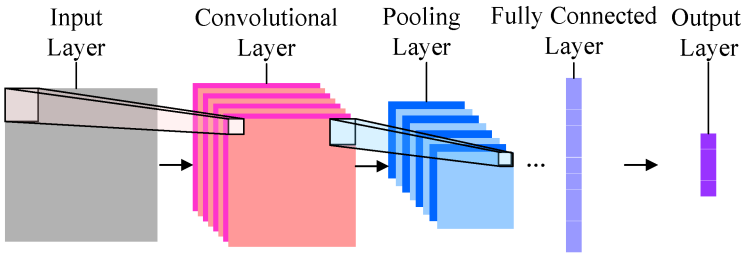


Figure 2: Typical CNN Neural Network from computer vision https://www.mdpi.com/2078-2489/7/4/614

A Convolutional Neural Network often comes with a set of neural network layers such as Convolutional Layer with filters to compute specific features e.g. compute the probability of being a edge, Pooling Layer which extracts specific feature from the computed values (which could be interpreted as probability) from Convolutional Layer e.g. edge in terms of image recognition.

The CNN is widely used in computer vision due to the underlying data structure of pictures. The typical property of such structure include conformity to contingent values, locality, sparsity, etc. In the later experiment, I will use CNN to extract the feature of a formula due to the fact that CNFs from a specifci domain often have similar convolutional structure.

3.2 Deep Q learning

Note The reinforcement learning theory can be very intriguing and here is the RL bible to obtain an introductory understanding of RL [7].
Deep Q learning is an extension of Q learning which is a tabular dynamic programming algorithm. The Q learning formula is as follows where Q is a state action function, s is a state, a is an action, r is reward , α is the learning rate, λ the discount factor and t is the time stamp. This formula is a derived form of the more general Bellman equation from value function.

$$Q^{next}(s_t, a_t) = (1 - \alpha)Q^{old}(s_t, a_t) + \alpha(r_t + \lambda \max_a Q(s_{t+1}, a))$$

4 Study on Solving Relative Induction more efficiently to improve PDR/IC3

From the paper published in 2011 on PDR algorithm, the researcher pointed out the key insight from A. Bradley due to his original work on IC3 which is a pioneering algorithm on PDR/IC3 (the name are sometimes interchangeably), which is an improvement in the SAT queries to help learn short clauses when solving inductive generalization it the procedure "SolveRelative" in the paper.

4.1 Relative Inductiveness and its importance in $PDR/IC3$ to find short clauses

a cube s is inductive relative to P if the following holds, T is the transition relation, P is the program state space.

$$P \wedge \neg s \wedge T \wedge s \ is \ UNSAT$$

This can be seen as an improvement on the following SAT queries in terms of the shorter size of the cube s than s'.

$$P \wedge \wedge T \wedge s' \ is \ UNSAT$$

In addition the following invariant has to be enforced as well, where $Init$ is the initial state.

$$Init \wedge s(s') \ is \ UNSAT$$

Finding a s' can be done relatively efficient by extracting the UNSAT core from the refutation proof generated by SAT solver. However, the clause $\neg s'$ learned

can be as large as the size of the state.
The clause learned from the first SAT query can be much shorter as a fact that when s becomes weaker (shorter) $\neg s$ can be a stronger constraint over the sates.

4.2 Deep Q learning to learn short clauses

To formalize the Reinforcement learning part of the problem, the input would be a counter example $\neg s_k$ found in step k. Another input is the learned formulae sequence (or the last several formulae). The output is a cube s and the clause of $\neg s$ is inductive relative to the current frame P.
I suggest to use a RNN to encode the transition formula T and the current frame F_k as $RNN0(T, F_k)$, which can be combined with statistics from the history of its predecessors using another RNN to form the input feature vector. The first RNN0 neural network is used to extract the structural patter from CNF formulae and also provide a uniform interface for formulae of variable length. I implement it as a RNN (GRU, gated rectified unit) to encode only the statistics of the SAT solving history (no RNN0) and it turned out to be enough to efficiently solve the two problems given to it. The RNN network is used to draw information from previous computations. Therefore the agent is able to learn the locality property of the formula and previous experience of tackling formulae of similar overall structure. The former short term information or locality property can be obtained by using any RNN structure. The latter reasoning from long term memory can be achieved by using a LSTM(Long Short Term Memory) unit of GRU (which is faster). The input feature vector can then be passed to a convolutional Deep Q neural network to compute the Q value for different actions on current state. The convolution neural network is used to capture the locality and repeated pattern of the SAT formula presented. Also many dense fully connected layer is added towards the output to enable agent to reason about complicated relations.

4.3 Evaluation

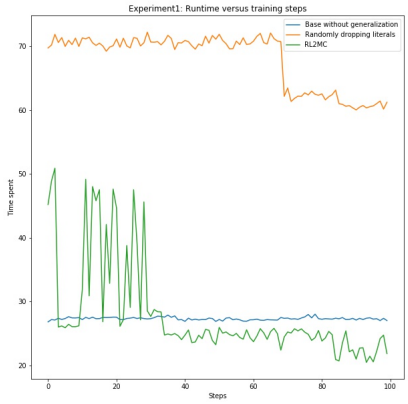


Figure 3: Experiment on problem BooleanIncrementer (Not Safe)

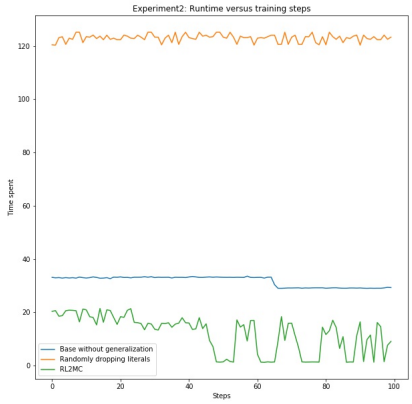


Figure 4: Experiment on problem IncrementerOverFlow (Safe)

Figure 3 shows the CPU run time as time steps change. The Base model without generalization (Base Model) spent shortest time at the beginning and stayed on that amount of time through out the end. The model which randomly drops literals to learn short clauses (Random Model) is the most expensive algorithm and very unstable. Finally the model with deep Q reinforcement learning (RL2MC) spent medium amount of time at the beginning but improved as time goes and it became the most efficient algorithm after 37 steps of training. It kept improving and its record shortest time is around 1/3 of the Base Model and its average time spent after 50 steps is around half of the time spent for Base Model.
Figure 4 showed even more promising result for RL2MC. The Random model is the worst and is around 100 times as slow as the RL2MC Model after 50 rounds. The RL2MC started with the shortest amount of time and became much better as time goes. Eventually it tends to converge to less than 2 seconds. This is a huge gain against the Base model which stays at around 33 throught the training.

In summary after the first few steps, the RL2MC beat the other two algorithms by a large margin and RL2MC kept improving. Though RL2MC still did not converge after 100 steps, but the pattern shows that RL2MC tends to beat Base Model by order of 10 and Random Model by order of 100. This is a huge improvement considering the simple input feature vector passed to the RL agent.

Note The time spent training the RL2MC agent is deducted when evaluating the model performance, however, the time spent on SAT queries are not. To obtain insightful comparison, the time spent for Random Model genreating random numbers is also deducted. The reason that training time is deducted is the training time is dependent on the computing resources and once trained successfully it will not affect the performance at all. Or Asymptotically, the training time does not matter. Actually the whole algorithm RL2MC completed Experiment2 the earliest.

5 conclusion

Heuristics in practice can reduce a specific set of problems to decidable or efficiently computable class. However, there is currently no efficient approach to automate heuristics searching other than exhaustive search the heuristics space. Moreover, classics heuristics reasoning scheme is ad hoc and no generalization can be directly applied to another set of similar problems other than trial and failure. In this paper, I propose to use reinforcement learning technique to automate heuristics search on specific set of industrial problems. Such idea is experimented on learning short clauses from relative induction in PDR/IC3 algorithms. The evaluation showed tremendous improvement on extracting short clauses efficiently compared to best effort exhaustive search as is currently widely

implemented. This approach of using reinforcement learning technique to automate heuristics search can be extended to other undecidable/non-efficiently computable problems in model checking community and can be easily paralleled.

References

[1] Aaron R. Bradley. Sat-based model checking without unrolling. In Ranjit Jhala and David Schmidt, editors, *Verification, Model Checking, and Abstract Interpretation*, pages 70–87, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[2] Aaron R. Bradley. Understanding ic3. In Alessandro Cimatti and Roberto Sebastiani, editors, *Theory and Applications of Satisfiability Testing – SAT 2012*, pages 1–14, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[3] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[4] Michael O'Gordon Duff and Andrew Barto. *Optimal Learning: Computational procedures for Bayes-adaptive Markov decision processes*. PhD thesis, University of Massachusetts at Amherst, 2002.

[5] Niklas Een, Alan Mishchenko, and Robert Brayton. Efficient implementation of property directed reachability. In *Proceedings of the International Conference on Formal Methods in Computer-Aided Design*, FMCAD '11, pages 125–134, Austin, TX, 2011. FMCAD Inc.

[6] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[7] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. 2018.