Reinforcement Learning State Space Properties for Model Checking of Interlockings*

Ben Lloyd-Roberts[1], Phil James[1], and Michael Edwards[1]

Swansea University, Swansea, UK
{ben.lloyd-roberts, p.d.james, michael.edwards}@swansea.ac.uk

Abstract. Railway interlockings serve as safety layer within railway signalling systems by ensuring proposed signalling requests are safe given the current state of the railway. As a vital part of any railway signalling system, interlockings are critical systems developed with the highest safety integrity level (SIL4) according to the CENELEC 50128 standard. The application of formal methods, in particular model-checking, in order to verify interlockings operate correctly is well established within academia and is beginning to see real applications in industry. However, the uptake of formal methods research within the UK rail industry has yet to make a substantial impact due to current approaches often producing false positives that require manual analysis during verification. Here, it is accepted that so-called invariants can be added to reduce the number of such false positives, however automatically computing these invariants remains a challenge. In this work we present a first approach illustrating how reinforcement learning can be used to learn properties of a state space for a given railway interlocking. Understanding how a verification problem can be formulated as one where machine learning can be successfully applied is the first step towards automated learning of invariants.

Keywords: Reinforcement Learning · Interlocking · Model Checking.

1 Introduction

Interlockings serve as a filter or 'safety layer' between inputs from operators, such as route setting requests, ensuring proposed changes to the current railway state avoid safety conflicts. As a vital part of any railway signalling system, interlockings are critical systems regarded with the highest safety integrity level (SIL4) according to the CENELEC 50128 standard. The application of model-checking to Ladder Logic programs in order to verify interlockings is well established within academia and is beginning to see real applications in industry. Bounded model-checking (BMC) is an efficient means of verifying a system through refutation. Given an abstracted model of a target system M and some safety properties ϕ, BMC searches for counterexamples up to some precomputed bound k. Search terminates when either an error trace is produced or the bound k is reached.

* Supported by Siemens Mobility UK & EPSRC

Determining this completeness threshold to sufficiently cover all states is often computationally intractable given it's true value will depend on model dynamics and size of the search space. Additionally, the k-induction rule, which checks if the property ϕ holds inductively for k sequences of states, is constrained to acyclic graphs for guarantees of completeness.

An invariance rule allows us to establish an invariant property ψ which holds for all initial states and transitions up to a given bound. Invariants may hold for sub-regions of the state space, meaning complete coverage isn't necessary to learn them. Supporting k-induction with strengthening invariants helps reduce the overall search space for bounded model checking, proving that invariant aspects of the system need not be considered. This can help filter cases where false negative counter examples are triggered by unreachable states. Generating sufficiently strong invariants is a non-trivial process given their construction is heavily program dependent. Usually such invariants require domain knowledge, typically devised by engineers responsible for the program implementation.

We infer two principle challenges to address. First, formulating theoretical and practical frameworks to represent invariant finding as a reinforcement learning task. Second, devising an appropriate probabilistic strategy for identifying learnable state space properties within those frameworks. With the ultimate aim of adapting our approach to identify ladder logic invariants, we present a set of preliminary results illustrating the potential for RL schemes to maximise state space coverage via learning progressively larger values for the recurrence reachability diameter.

2 BMC & Program Invariants

- Not sure if this section is needed?

3 Reinforcement Learning

RL is a popular ML paradigm with a demonstrably impressive capacity for modelling sequential decision making problems as the optimal control of some incompletely-known Markov Decision Process (MDP), known as the environment [9]. Given a permitted set of actions to perform over a series of discrete time steps t, software agent(s) are trained to interact with and observe changes in the environment based on intermittent reward signals. Over a process of accelerated trial-and-error a function, or policy π, is learned mapping states to optimal actions likely to return the greatest cumulative future reward. Interactions with the environment can be characterised as continuous or episodic, depending on constraints of the learned task.

Tabular reinforcement learning has existed for decades through dynamic programming, Monte-Carlo methods, and temporal difference learning. Resurgence in RL research over the last decade, applied to games [7,8,10], robotics [1,2] and

operations research [4], can be attributed to the fusion of a existing value/policy iteration methods with powerful approximate learning techniques following the advent of deep learning [3].

Subsequent popularity in deep reinforcement learning (DRL) after breakthrough performance of Deep Q-Learning on a set of gaming benchmarks [6] gave rise to developments in Actor-Critic methods, combine - learning both the policy which dictates behaviour (actor) and the value functions which estimate the quality of a given state (critic). What are the known limitations? Prior to actor-critic with experience replay, On policy nature of actor critic meant previous experiences from old policy iterations were forgotten, thus biasing behaviour to the most recent model updates and introducing sample inefficiency.

Probabilistic learning is unlikely to provide us with guarantees of completeness so there's not much scope to replace formal methods but can be used to supplement them. For purposes of learning heuristics, we would ideally like to maximise the information gathered, so that there is more data from which to identify patterns. With respect to invariant finding, we aim to maximise state space, or environment, coverage. Fortunately existing works have illustrated the efficacy of RL methods in learning such heuristics over large graph structures (heuristics paper), distributing the task among concurrent workers for accelerated performance (A3C and all distributed GPU cluster papers) and prioritising exploration in unfamiliar environments (count based-exploration, maximising network entropy, coverage via curiosity).

Reinforcing this behaviour requires motivating agents via a sparse reward signal with potentially strong temporal dependence. In light of this, we influence agents to pursue the longest loop free path where a reward scheme positively rewards sequences of novel observations. Inversely, negative rewards are issued for repeated observations within a training episodes.

We select asynchronous advantage actor-critic [5] which estimates both the value function and behaviour policy when exploring an environment. The asynchronous nature of the algorithm facilitates distributed exploration of state-action pairs via separate workers with a shared global model of their environment.

3.1 Environment Generation

Given exhaustive search of large state spaces is often computationally intractable, we generate a set of ladder logic programs where the number of reachable states and recurrence reachability diameter are known. Using the well established pelican crossing example as a base template, we introduce a single contact and coil with each additional rung during program generation. This way a constrained yet predictable pattern of growth is introduced. If $|S(p_i)|$ represents the number of reachable states for a program p_i, a subsequently generated program p_{i+1} with one additional rung, has $2|S(p_i)| + 1$ reachable states. Through a series of training runs on each environment we record the number of states observed by workers to gauge the overall state space coverage.

4 Learning Framework

We implement our environment as the set of all reachable states produced by executions of some arbitrary ladder logic program. Workers transition between states via changes in valuation of the action set A, which comprises all contact variables in the LL program. Each state is characterised by the observation space S, the union of contacts and coils under their valuation following the latest action, i.e change in A.

For practicality, each environment has an associated max number of episodes $T_{\max}$ to bound training runs. Additionally we utilise two forms of early termination to avoid superfluous interaction with the environment. First, if the performance curve of a given agent converges to some local minima. Second, if all reachable states have been observed at least once. Given agents accumulate experience within separate environment instances, they update shared sets of observations and rewards intermittently. This update frequency depends on both the number of workers and precomputed size of the environment.

5 Results

Stochasticity of environment and network initialisation causes some variability in results.

Increased training time does not guarantee improved results. We empirically observe three scenarios: (*i*) Performance curves converge to a reward bounded by the longest loop free path; (*ii*) Performance curves converge to a suboptimal reward; (*iii*) Performance improves linearly before collapsing to some suboptimal reward.

Given the A3C algorithm requires workers to asynchronously update their shared network every $T_{\max}$ steps or on episode termination, larger values for $T_{\max}$ consolidate more information regarding worker trajectories before applying gradient updates to their local network. Therefore we observe a strong dependence on this parameter both in terms of increasing the k bound when introducing workers to larger environments. Prior to random episode initialisation and fine-tuning network update frequencies, workers seldom covered 100% of the smaller environments.

6 Conclusion & Future Work

A3C shows some promising preliminary results but is clearly limited in its capacity to scale. The algorithm shows proclivity to collapse following small network adjustments in previously unexplored regions of large environments.

Introduce experience replay for distributed learning to improve on-policy bias and sample efficiency. Additional algorithms to implement for scalability - we can guarantee the increase of action spaces when transitioning to industry scale ladder logic programs where the number of contacts are known. It may be worth exploring DDPG to improve learning solely based on the observation space.

Table 1. Initial results applying A3C learning over progressively larger environments.

Environment Metrics			Training Metrics			
States	**Reachable States**	**Actions**	**Observed K**	**States Observed**	**Coverage**	**Total Episodes**
2^{12}	7	2	6	7	100	1.00E+04
2^{14}	15	4	14	15	100	1.00E+04
2^{16}	31	8	28	31	100	1.00E+04
2^{18}	63	16	48	63	100	1.00E+05
2^{20}	127	32	33	127	100	1.00E+05
2^{22}	255	64	76	255	100	1.00E+04
2^{24}	511	128	49	377	100	1.00E+05
2^{26}	1023	256	306	1023	100	3.00E+05
2^{28}	2047	512	538	2047	100	3.00E+05
2^{30}	4095	1024	1418	4084	99.731	3.00E+05
2^{32}	8191	2048	1712	7907	96.532	3.00E+05
2^{34}	16383	4096	1498	15654	95.550	3.00E+05
2^{36}	32767	8192	2879	27752	84.694	3.00E+05
2^{38}	65535	16384	1969	58373	89.071	3.00E+05
2^{40}	131071	32768	2692	108638	82.884	2.00E+05
2^{42}	262143	65536	1406	5199317	76.033	3.00E+05
2^{44}	524287	131072	1782	325781	62.137	2.00E+05
2^{46}	1048575	262144	1593	671645	64.053	3.00E+05
2^{48}	2097151	524288	1598	1206867	57.547	3.00E+05
2^{50}	4194303	1048576	2527	1739939	41.48	1.50E+05

Intrinsic motivation for exploration.

IMPALA to improve both sample efficiency over A3C and robustness to network architectures and hyperparameters. Use of LSTM also improves performance given GPU acceleration is maximised on larger batch updates. A3C in our setting experiences variable episodic updates.

References

1. Bloesch, M., Humplik, J., Patraucean, V., Hafner, R., Haarnoja, T., Byravan, A., Siegel, N.Y., Tunyasuvunakool, S., Casarini, F., Batchelor, N., et al.: Towards real robot learning in the wild: A case study in bipedal locomotion. In: Conference on Robot Learning. pp. 1502–1511. PMLR (2022)
2. Gu, S., Holly, E., Lillicrap, T., Levine, S.: Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In: 2017 IEEE international conference on robotics and automation (ICRA). pp. 3389–3396. IEEE (2017)
3. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. nature **521**(7553), 436–444 (2015)
4. Mazyavkina, N., Sviridov, S., Ivanov, S., Burnaev, E.: Reinforcement learning for combinatorial optimization: A survey. Computers & Operations Research **134**, 105400 (2021)

5. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T.P., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning (2016)
6. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 (2013)
7. Schaul, T., Quan, J., Antonoglou, I., Silver, D.: Prioritized experience replay. arXiv preprint arXiv:1511.05952 (2015)
8. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al.: Mastering the game of go with deep neural networks and tree search. nature **529**(7587), 484–489 (2016)
9. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press (2018)
10. Vinyals, O., Babuschkin, I., Czarnecki, W.M., Mathieu, M., Dudzik, A., Chung, J., Choi, D.H., Powell, R., Ewalds, T., Georgiev, P., et al.: Grandmaster level in starcraft ii using multi-agent reinforcement learning. Nature **575**(7782), 350–354 (2019)