Towards Reinforcement Learning of Invariants for Model Checking of Interlockings

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

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

Abstract. 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 first steps towards using reinforcement learning to learn properties of a state space.

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. As early as 1995, Groote et al. [11] applied formal methods to verify an interlocking for controlling the Hoorn-Kersenbooger railway station. They conjecture the feasibility of verification techniques as means of ensuring correctness criteria on larger railway yards. In 1998, Fokkink and Hollingshead [8] suggested a systematic translation of Ladder Logic into Boolean formulae. Newer approaches to interlocking verification have also been proposed in recent years [6, 7, 14]. This includes work by Linh et al. which explores the verification of interlockings written in a similar language to Ladder Logic using SAT-based model checking. After two decades of research, academic work [17, 19] has shown that verification approaches for Ladder Logic can indeed scale; in an industrial pilot, Duggan et al. [14] conclude: "Formal proof as a means to verify safety has matured to the point where it can be applied for any railway interlocking system." In spite of this, such approaches still lack widespread use within the Rail industry.

Principally our work aims to address one of the issues hindering its uptake, by removing the need for manual analysis of false positive error traces produced during verification.

In this work, we firstly formulate the theoretical and practical frameworks to represent interlocking verification as a goal-orientated reinforcement learning task and secondly, devise an appropriate strategy for learning invariants within those frameworks.

2 Preliminaries

We now briefly discuss both model checking of railway interlockings and reinforcement learning. For further details we refer the reader to [17, 19] and [23] respectively.

2.1 Bounded Model Checking and Invariants

Model checking is a formal verification technique stemming from the need to systematically check whether certain properties hold for different configurations (states) of a given system. Given a finite transition system T and a formula F, model checking attempts to verify that $s \vdash F$ for every system state $s \in T$, such that $T \vdash F$.

The model checking process culminates in verification results being generated. Properties which hold for all tested states produce a 'safe' output. In the event a state is found to violate any specified properties, that is $s \nvdash F$, a counter example trace is provided by the model checker indicating which state(s) caused the violation. Results may also indicate that the model, property formulation or simulation process are insufficient for verification and therefore require further refinement.

Primary limitations of several model checking solutions indicate that verification can fail due to over approximation of the model being checked, typically when using techniques such as inductive verification. Such inductive verification checks to see if a given state satisfies some condition but does not consider whether these states which violate the same safety condition are reachable by the system. These false positive counter examples often require manual inspection by an experienced engineer. Here, one solution that is proposed [1] is to introduce so-called invariants to suppress false positives. Invariants are properties that hold for sub-regions of the state space. The aim is to introduce invariants that help bound the region of reachable states when model checking. However generating sufficiently strong invariants automatically is a complex task, one which has received considerable attention in academic literature. From software engineering techniques [2, 4] to hybrid methods incorporating machine learning [9], researchers have proposed various approaches to invariant finding with varying degrees of success.

2.2 Reinforcement Learning

Reinforcement Learning (RL) is a popular machine learning paradigm with demonstrably impressive capacity for modelling sequential decision making problems as the optimal control of some incompletely-known Markov Decision Process (MDP) [30]. Given a permitted set of actions, performable over a series of discrete time steps, software agents learn a function mapping MDP states to optimal actions likely to maximise cumulative reward signals. Interactions with the environment can be characterised as continuous or episodic, depending on constraints of the learned task.

Resurgence in RL research over the last decade has seen applications in games [26, 29, 31], robotics [3, 12] and operations research [22] and can be attributed to the fusion of existing methods [33] with powerful approximate learning techniques [20] following the advent of deep learning [24]. Subsequent popularity in deep reinforcement learning (DRL) gave rise to improvements of historic methods [27]. Actor-Critic methods, combining policy learning [18] and value function approximation saw particular successes in state-of-the-art performance [28].

Probabilistic learning is unlikely to provide guarantees of completeness, but can be used to supplement formal methods, such as model checking. For purposes of learning heuristics, we would like to maximise the information gathered, so that there is more data from which to identify patterns. With the aim of learning invariant properties that hold across states, state space coverage would be ideally maximised. Fortunately existing works have illustrated the efficacy of RL methods in learning such heuristics over large graph structures [21], distributing learning for accelerated performance [15] and prioritising exploration in unfamiliar environments [10, 13, 25].

Throughout this work we have used asynchronous advantage actor-critic (A3C) to estimate both the value function and behaviour policy while exploring an environment. The asynchronous nature of the algorithm facilitates distributed exploration of state-action pairs via separate workers with a shared global policy network.

3 Mapping Formal Methods to Reinforcement Learning

For this work, we have concentrated on trying to produce invariants for the approaches taken by Kanso et al [19] and James et al [17]. Here we include their model of ladder logic based railway interlocking programs as we use this as a basis for defining an learning environment. The semantics of a ladder logic program is defined as a function of input and output variable valuations: $[\psi] : V_I \times V_C \to V_C$. Therefore $\psi(\mu_I, \mu_C) = \mu_{C'}$, where

$$\mu_{C'}(c_i) = [\psi](\mu_I, \{c_i, ..., c_n\}, \{c'_1, ..., c'_{i-1}\}) \quad (1)$$

$$\mu_{C'}(c) = \mu_C \textbf{ if } c \notin \{c_1, ..., c_n\} \quad (2)$$

A Ladder Logic labelled transition system $LTS(\psi)$ is defined as the three tuple $(V_C, \to, V_0)$:

- $\mu_C \rightarrow \mu_{C'}$ *iff* $[\psi](\mu_C, \mu_I) = \mu_{C'}$
- $V_0 = \{\mu_C \mid init(\mu_C)\}$,

where the function $init()$ produces the initial valuation of variables μ_C, setting all as false.

We now define the finite Markov Decision Process (MDP), or environment $\mathcal{E}$ used to represent Ladder Logic program. A Ladder Logic MDP $M(\psi)$ is a five tuple $\langle S, \mathcal{A}, P_a(s, s'), R_a(s, s'), \gamma \rangle$, where

- $S = V_I \cup V_C$, observation space to represent the MDP state at discrete time steps.
- $\mathcal{A} = V_I$, describes the action space; a set of formally defined actions which change the observation space
- $P_a(s, s') = Pr(s_{t+1} = s' | s_t, a_t)$, describing the likelihood of observing state s_{t+1} given action a_t taken from state s_t
- $R_a(s, s')$ is a reward function fed back to the agent at each time step
- γ is a discount scalar applied to the reward estimates for future time steps.

Subsequently the environment unfolds as a set of reachable states for the respective Ladder Logic program. As workers improve their value estimates according to the reward function, a balance is maintained between stochastic action sampling and predictions from the policy network.

Given invariant properties hold for some subregion of program states, we aim to reinforce exploration to maximise MDP coverage. In light of this, we influence agents to pursue the longest loop free path, or max k value for BMC. Consequently we design a reward scheme which positively rewards sequences of novel observations. Inversely, negative rewards are issued for repeated observations within a training episodes. Workers are initialised with separate environment instances to accumulate experience independently. A global set of observations is asynchronously updated by workers periodically to compile shared experiences.

For practicality, each environment has an associated max number of episodes $T_{\max}$ to constrain runtime. We utilise two forms of early termination to avoid superfluous training. First, if worker performance curves converges to some local minima. Second, if all reachable states have been observed at least once. To aid exploration, on episode resets we randomly initialise workers in some previously visited state, as demonstrated in [10].

4 Results

We now present a set of results from applying our approach to a series of generated Ladder Logic programs, modelled as learning environments.

4.1 Environment Generation

Given exhaustive search of large state spaces is often computationally intractable, we have generated a set of ladder logic programs where the number of reachable states and recurrence reachability diameter are known. This has enabled us to analyse the performance of our approach against well understood state spaces. Using existing models of ladder logic structures as a base template [17], we derive progressively larger programs by sequentially introducing additional rungs. This way a constrained yet predictable pattern of growth is devised. 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.

4.2 Discussion

Preliminary results applying our approach to a number of generated programs are outlined in Table 1. 'Actions' referenced in column 3 refer to the number of possible assignments over input variables in each Ladder Logic program. 'K', refers to the greatest number of steps taken before repeating observations n the environment, across all workers.

Coverage metrics are expectedly maximised for environments with a small number of reachable states with acceptable levels of coverage for programs with more than 1e5 states. Interestingly, we observed longer training durations occasionally increased coverage beyond a certain threshold. It is possible workers learn an optimal search strategy within a subregion of the state space. Additionally, performance in terms of max k and states reached increased by approx. 5% when decreasing the total number of episodes from 3e5 to 1.5e5 episodes. This may be a product of random episode initialisation spawning workers in more desirable states where stochastic action sampling happened to lead to unfamiliar subregions of the environment.

Performance plots illustrating the cumulative reward which failed to maximise coverage often increased linearly before collapsing to some suboptimal reward. This may be due to tendencies for large network updates to shift the network gradients into a bad local minima, from which performance does not recover within the allotted training duration. The on-policy nature of actor critic means trajectories generated via an old policy are no longer sampled during minibatch updates for the current policy, thus biasing behaviour to the most recent model updates and introducing sample inefficiency. Adding experience replay [32] may help avoid this in future applications

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. We found the most significant improvements to performance in terms of coverage metrics and increasing the k bound when introducing workers to larger environments, was lower update frequencies and random start state initialisation. Prior to these adjustments workers, irrespective of their number, seldom covered 80% of most smaller environments. Similarly, for the largest environment with 2^{50} states, coverage improved from 3.2% to 41.48%

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

Environment Metrics			Training Metrics			
States	**Reachable States**	**Actions**	K	**States Reached**	**Coverage**	**Total Episodes**
2^{12}	7	2^{1}	6	7	100	1.00E+04
2^{14}	15	2^{2}	14	15	100	1.00E+04
2^{16}	31	2^{3}	28	31	100	1.00E+04
2^{18}	63	2^{4}	48	63	100	1.00E+05
2^{20}	127	2^{5}	33	127	100	1.00E+05
2^{22}	255	2^{6}	76	255	100	1.00E+04
2^{24}	511	2^{7}	49	377	100	1.00E+05
2^{26}	1023	2^{8}	306	1023	100	3.00E+05
2^{28}	2047	2^{9}	538	2047	100	3.00E+05
2^{30}	4095	2^{10}	1418	4084	99.731	3.00E+05
2^{32}	8191	2^{11}	1712	7907	96.532	3.00E+05
2^{34}	16383	2^{12}	1498	15654	95.550	3.00E+05
2^{36}	32767	2^{13}	2879	27752	84.694	3.00E+05
2^{38}	65535	2^{14}	1969	58373	89.071	3.00E+05
2^{40}	131071	2^{15}	2692	108638	82.884	2.00E+05
2^{42}	262143	2^{16}	1406	5199317	76.033	3.00E+05
2^{44}	524287	2^{17}	1782	325781	62.137	2.00E+05
2^{46}	1048575	2^{18}	1593	671645	64.053	3.00E+05
2^{48}	2097151	2^{19}	1598	1206867	57.547	3.00E+05
2^{50}	4194303	2^{20}	2527	1739939	41.48	1.50E+05

5 Conclusion & Future Work

In this paper we have applied a basic asynchronous deep reinforcement learning method to maximise program state coverage, motivated by a reward scheme . some promising preliminary results but limited in its capacity to scale across large observation spaces.

In light of our findings, we aim to improve several aspects of our approach, predominantly concerning learning stability, sample efficiency and training speed. Experience replay for distributed learning may improve on-policy bias and sample efficiency.

The low dimensionality of our state spaces representation may allow us to introduce count-based exploration models to dampen the reward issued for states repeatedly observed [25]. Intrinsic motivation has also illustrated successes in environment exploration [16].

Applying IMPALA [5] to improve both sample efficiency over A3C and robustness to network architectures and hyperparameter adjustments. The adoption of a Long Short-Term Memory model (LSTM) also improves performance given GPU acceleration is maximised on larger batch updates. [1]

[1] **Acknowledgment:** We thank Tom Werner and Andrew Lawrence at Siemens Mobility UK & EPSRC for their support in these works.

References

1. Awedh, M., Somenzi, F.: Automatic invariant strengthening to prove properties in bounded model checking. In: 2006 43rd ACM/IEEE Design Automation Conference. pp. 1073–1076 (2006). https://doi.org/10.1145/1146909.1147180
2. Bensalem, S., Lakhnech, Y., Saidi, H.: Powerful techniques for the automatic generation of invariants. In: International Conference on Computer Aided Verification. pp. 323–335. Springer (1996)
3. 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)
4. Case, M.L., Mishchenko, A., Brayton, R.K.: Automated extraction of inductive invariants to aid model checking. In: Formal Methods in Computer Aided Design (FMCAD'07). pp. 165–172. IEEE (2007)
5. Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., Legg, S., Kavukcuoglu, K.: Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures (2018)
6. Fantechi, A., Fokkink, W., Morzenti, A.: Some trends in formal methods applications to railway signaling. Formal methods for industrial critical systems: A survey of applications pp. 61–84 (2012)
7. Ferrari, A., Magnani, G., Grasso, D., Fantechi, A.: Model checking interlocking control tables. In: FORMS/FORMAT 2010, pp. 107–115. Springer (2011)
8. Fokkink, W., Hollingshead, P., Groote, J., Luttik, S., van Wamel, J.: Verification of interlockings: from control tables to ladder logic diagrams. In: Proceedings of FMICS. vol. 98, pp. 171–185 (1998)
9. Garg, P., Neider, D., Madhusudan, P., Roth, D.: Learning invariants using decision trees and implication counterexamples. ACM Sigplan Notices **51**(1), 499–512 (2016)
10. Gordillo, C., Bergdahl, J., Tollmar, K., Gisslén, L.: Improving playtesting coverage via curiosity driven reinforcement learning agents. arXiv preprint arXiv:2103.13798 (2021)
11. Groote, J.F., van Vlijmen, S.F., Koorn, J.W.: The safety guaranteeing system at station hoorn-kersenboogerd. In: COMPASS'95 Proceedings of the Tenth Annual Conference on Computer Assurance Systems Integrity, Software Safety and Process Security'. pp. 57–68. IEEE (1995)
12. 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)
13. Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor (2018)
14. Haxthausen, A.E., Bliguet, M.L., Kjær, A.A.: Modelling and verification of relay interlocking systems. In: Monterey Workshop. pp. 141–153. Springer (2008)
15. Hoffman, M., Shahriari, B., Aslanides, J., Barth-Maron, G., Behbahani, F., Norman, T., Abdolmaleki, A., Cassirer, A., Yang, F., Baumli, K., et al.: Acme: A research framework for distributed reinforcement learning. arXiv preprint arXiv:2006.00979 (2020)
16. Houthooft, R., Chen, X., Duan, Y., Schulman, J., Turck, F.D., Abbeel, P.: Vime: Variational information maximizing exploration (2017)

17. James, P., Lawrence, A., Moller, F., Roggenbach, M., Seisenberger, M., Setzer, A., Kanso, K., Chadwick, S.: Verification of solid state interlocking programs. In: International Conference on Software Engineering and Formal Methods. pp. 253–268. Springer (2013)
18. Kakade, S.M.: A natural policy gradient. Advances in neural information processing systems **14** (2001)
19. Kanso, K., Moller, F., Setzer, A.: Automated verification of signalling principles in railway interlocking systems. Electronic Notes in Theoretical Computer Science **250**(2), 19–31 (2009)
20. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. nature **521**(7553), 436–444 (2015)
21. Manchanda, S., Mittal, A., Dhawan, A., Medya, S., Ranu, S., Singh, A.: Learning heuristics over large graphs via deep reinforcement learning. arXiv preprint arXiv:1903.03332 (2019)
22. Mazyavkina, N., Sviridov, S., Ivanov, S., Burnaev, E.: Reinforcement learning for combinatorial optimization: A survey. Computers & Operations Research **134**, 105400 (2021)
23. 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)
24. 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)
25. Ostrovski, G., Bellemare, M.G., van den Oord, A., Munos, R.: Count-based exploration with neural density models (2017)
26. Schaul, T., Quan, J., Antonoglou, I., Silver, D.: Prioritized experience replay. arXiv preprint arXiv:1511.05952 (2015)
27. Schulman, J., Levine, S., Moritz, P., Jordan, M.I., Abbeel, P.: Trust region policy optimization (2017)
28. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms (2017)
29. 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)
30. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press (2018)
31. 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)
32. Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., de Freitas, N.: Sample efficient actor-critic with experience replay (2017)
33. Watkins, C.J., Dayan, P.: Q-learning. Machine learning **8**(3), 279–292 (1992)