Todo list

Towards Reinforcement Learning of Invariants for Model Checking of Interlockings

Ben Lloyd-Roberts, Phillip James, Michael Edwards
Computational Foundry, Swansea University
Swansea, United Kingdom
Email: {ben.lloyd-roberts, p.d.james, michael.edwards}@swansea.ac.uk

***Abstract*—The application of formal methods, in particular model checking, 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, properties which hold for the entirety or a substantial subregion of the search space, can help reduce the number of such false positives. Invariants are often bespoke, manually designed by engineers making their automatic generation a challenge. In this work we present first steps towards using reinforcement learning to navigate state space representations of ladder logic programs and generate a dataset of state sequences from which invariants could be mined.**

Results and what they suggest

1. Introduction

Interlockings serve as a filter or 'safety layer' between inputs from railway 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 (LLPs) in order to verify interlockings is well established within academia and is beginning to see real applications in industry.

In this work we build upon existing techniques in modelling LLP dynamics. First we propose a theoretical and practical framework for navigating state space representations of LLPs as a goal-orientated reinforcement learning task. Paired with selective reset logic during the training phase, see section 3.2, software agents are incentivised to maximise state space coverage by pursuing sequences of unobserved states, i.e the max acyclic subgraph, for a given interlocking program. During the agent's exploration phase state sequences are generated, from which we eventually hope to mine invariant properties.

Elaborate on invariant finding problem

2. Preliminaries

We now briefly discuss model checking of railway interlockings and reinforcement learning. For further details we refer the reader to [6], [7] and [8] respectively.

2.1. Ladder Logic & Interlockings

Brief overview of ladder logic structure?

2.2. Verification in the Railway

Overview of verification attempts in railway domain

As early as 1995, Groote et al. [1] 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 [2] suggested a systematic translation of Ladder Logic into Boolean formulae. Newer approaches to interlocking verification have also been proposed in recent years [3], [4], [5]. 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 [6], [7] 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.

2.3. Bounded Model Checking and Invariants

Add specifics of BMC and interlockings

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 through refutation 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 infraction. Results may indicate that the model, property formulation or simulation process are insufficient for verification, necessitating 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 [9] is to introduce so-called invariants to suppress false positives. Invariants are properties that hold for subregions 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 [10], [11] to hybrid methods incorporating machine learning [12], researchers have proposed various approaches to invariant finding with varying degrees of success.

2.4. 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) [13]. This user defined *environment*, $\mathcal{E}$, comprises a set of unique states $\mathcal{S}$, a set of permitted actions from each state $\mathcal{A}(s)$, a function describing state transitions $f : S \times \mathcal{A} \rightarrow S^{+}$ and a scalar reward signal r_t. States, actions and rewards are measured over discrete time steps t, where simulations are summarised through the *trajectory*, $\tau = (s_1, a_1, r_1, s_2, a_2, r_2, ..., s_h, a_h, r_h)$, where h refers to the *horizon*, a time step beyond which rewards are no longer considered. Through repeated simulations, software agents aim to optimise a deterministic or stochastic behaviour function known as the policy, denoted $\pi(s)$ or $\pi(a|s)$ respectively, mapping MDP states to optimal actions expected to maximise cumulative rewards over time. We refer to the summary reward objective as the *expected return*, $\mathbb{E} = [G_t|S_t = s, A_t = a]$, an empirical average taken over future *discounted returns* from the current time step $G_t = \sum_{i=t}^{T} \gamma^{i-t} r_i$. Here, T refers to a terminal time step where simulation ends according to some stopping criteria. Thereafter, learning may conclude having achieved adequate performance or resume from some initial state following an environment reset. The discount factor $\gamma \in [0, 1]$ applies to successive rewards at each time step to help enumerate returns over a potentially infinite horizon.

Resurgence in RL research over the last decade has seen applications in games [14], [15], [16], robotics [17], [18] and operations research [19] and can be attributed to the fusion of existing methods [20] with powerful approximate learning techniques [21] following the advent of deep learning [22]. Subsequent popularity in deep reinforcement learning (DRL) gave rise to improvements of historic methods [23]. Actor-Critic methods, combining policy learning [24] and value function approximation saw particular successes in establishing state-of-the-art performance [25].

Probabilistic learning is unlikely to provide guarantees of completeness, but can be used to supplement formal methods, such as model checking, via learned heuristics. We posit an approach of information maximisation, collecting sufficient trajectories from which to identify patterns or sequences. With the aim of learning invariant properties that hold across states, state space coverage should be maximised.

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. [6] and James et al. [7]. Here we include their model of ladder logic based railway interlocking programs as we use this as a basis for defining an learning environment.

Explain why the LTS in James et al. [17] uses four elements (transition label) but our version doesn’t

3.1. Modelling Ladder Logic

The semantics of a ladder logic program (LLP) is defined as a function of input and output variable valuations

Describe 3-tuple for LTS

label : $[\psi] : V_I \times V_C \rightarrow V_C$

Distinguish between V_I and V_C better

Therefore $\psi(\mu_I, \mu_C) = \mu_{C'}$, where -

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

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

Fix transcription errors in ladder logic definition

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

$$\mu_C \rightarrow \mu_{C'} \textit{ iff } [\psi](\mu_C, \mu_I) = \mu_{C'} \quad (3)$$
$$V_0 = \{\mu_C \mid init(\mu_C)\}, \quad (4)$$

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

3.2. Ladder Logic Markov Decision Process

We now define the finite Markov Decision Process (MDP), or environment $\mathcal{E}$ used to represent the LLP. 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 LLP. As workers improve their value estimates according to the reward function, a balance is maintained between stochastic action sampling for exploration and best 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 Bounded Model Checking. Consequently we design a reward scheme which positively rewards sequences of novel observations. Inversely, negative rewards are issued for repeated observations within the same 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, i.e consecutive model updates result in no further improvement. Second, training ends in our artificially generated programs 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 [30].

4. Related Work

Here we briefly highlight key contributions within related literature, addressing the invariant finding problem for interlocking programs and contemporary RL strategies for environment exploration.

4.1. Invariant Finding

4.2. Reinforcement Learning

Existing works have illustrated the efficacy of RL methods in learning such heuristics over large graph structures [26], distributing learning for accelerated performance [27] and prioritising novelty when exploration unfamiliar environments [28], [29], [30].

5. Results

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

5.1. Environment Generation

Given exhaustive search of large state spaces is often computationally intractable, we have generated a set of LLPs 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 [7], 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.

5.2. Discussion

Elaborate on the efficacy of approach - is it actually promising?

Add results on some Loch Ness interlocking examples

Compare existing BCM approaches to learning heuristic?

Preliminary results applying our approach to a number of generated programs are outlined in Table **??**. 'Actions' referenced in column 3 refer to the number of possible assignments over input variables in each LLP. '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.

Fix scientific notation

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.

Include performance plots

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 [31] 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%

6. 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

Sentence?

. 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 [28]. Intrinsic motivation has also illustrated successes in environment exploration [32].

Applying IMPALA [33] 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.

Acknowledgments

We thank Tom Werner and Andrew Lawrence at Siemens Mobility UK & EPSRC for their support in these works.

References

[1] J. F. Groote, S. F. van Vlijmen, and J. W. Koorn, "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'*. IEEE, 1995, pp. 57–68.

[2] W. Fokkink, P. Hollingshead, J. Groote, S. Luttik, and J. van Wamel, "Verification of interlockings: from control tables to ladder logic diagrams," in *Proceedings of FMICS*, vol. 98, 1998, pp. 171–185.

[3] A. Fantechi, W. Fokkink, and A. Morzenti, "Some trends in formal methods applications to railway signaling," *Formal methods for industrial critical systems: A survey of applications*, pp. 61–84, 2012.

[4] A. Ferrari, G. Magnani, D. Grasso, and A. Fantechi, "Model checking interlocking control tables," in *FORMS/FORMAT 2010*. Springer, 2011, pp. 107–115.

[5] A. E. Haxthausen, M. L. Bliguet, and A. A. Kjær, "Modelling and verification of relay interlocking systems," in *Monterey Workshop*. Springer, 2008, pp. 141–153.

[6] K. Kanso, F. Moller, and A. Setzer, "Automated verification of signalling principles in railway interlocking systems," *Electronic Notes in Theoretical Computer Science*, vol. 250, no. 2, pp. 19–31, 2009.

[7] P. James, A. Lawrence, F. Moller, M. Roggenbach, M. Seisenberger, A. Setzer, K. Kanso, and S. Chadwick, "Verification of solid state interlocking programs," in *International Conference on Software Engineering and Formal Methods*. Springer, 2013, pp. 253–268.

[8] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," 2016.

[9] M. Awedh and F. Somenzi, "Automatic invariant strengthening to prove properties in bounded model checking," in *2006 43rd ACM/IEEE Design Automation Conference*, 2006, pp. 1073–1076.

[10] M. L. Case, A. Mishchenko, and R. K. Brayton, "Automated extraction of inductive invariants to aid model checking," in *Formal Methods in Computer Aided Design (FMCAD'07)*. IEEE, 2007, pp. 165–172.

[11] S. Bensalem, Y. Lakhnech, and H. Saidi, "Powerful techniques for the automatic generation of invariants," in *International Conference on Computer Aided Verification*. Springer, 1996, pp. 323–335.

[12] P. Garg, D. Neider, P. Madhusudan, and D. Roth, "Learning invariants using decision trees and implication counterexamples," *ACM Sigplan Notices*, vol. 51, no. 1, pp. 499–512, 2016.

[13] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[14] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[15] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[16] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[17] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 3389–3396.

[18] M. Bloesch, J. Humplik, V. Patraucean, R. Hafner, T. Haarnoja, A. Byravan, N. Y. Siegel, S. Tunyasuvunakool, F. Casarini, N. Batchelor *et al.*, "Towards real robot learning in the wild: A case study in bipedal locomotion," in *Conference on Robot Learning*. PMLR, 2022, pp. 1502–1511.

[19] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev, "Reinforcement learning for combinatorial optimization: A survey," *Computers & Operations Research*, vol. 134, p. 105400, 2021.

[20] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3, pp. 279–292, 1992.

[21] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[22] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[23] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," 2017.

[24] S. M. Kakade, "A natural policy gradient," *Advances in neural information processing systems*, vol. 14, 2001.

[25] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[26] S. Manchanda, A. Mittal, A. Dhawan, S. Medya, S. Ranu, and A. Singh, "Learning heuristics over large graphs via deep reinforcement learning," *arXiv preprint arXiv:1903.03332*, 2019.

[27] M. Hoffman, B. Shahriari, J. Aslanides, G. Barth-Maron, F. Behbahani, T. Norman, A. Abdolmaleki, A. Cassirer, F. Yang, K. Baumli *et al.*, "Acme: A research framework for distributed reinforcement learning," *arXiv preprint arXiv:2006.00979*, 2020.

[28] G. Ostrovski, M. G. Bellemare, A. van den Oord, and R. Munos, "Count-based exploration with neural density models," 2017.

[29] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," 2018.

[30] C. Gordillo, J. Bergdahl, K. Tollmar, and L. Gisslén, "Improving playtesting coverage via curiosity driven reinforcement learning agents," *arXiv preprint arXiv:2103.13798*, 2021.

[31] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, "Sample efficient actor-critic with experience replay," 2017.

[32] R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. D. Turck, and P. Abbeel, "Vime: Variational information maximizing exploration," 2017.

[33] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, and K. Kavukcuoglu, "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures," 2018.