Mining Invariants from State Space Observations

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

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

Abstract

The application of formal methods to verify railway systems operate correctly is well established within academia and is beginning to see real applications. However, many current approaches often produce false positive results that require lengthy manual analysis. It is accepted that so-called invariants, properties which hold for all states under which a system operates, can help reduce occurrences of false positives. However, automated deduction of these invariants remains a challenge. In this work, we use reinforcement learning to build a dataset of state observations and compute correlation coefficients between all variables composing a program, allowing proposals for candidate invariant properties.

1 Introduction

Model checking is a formal verification technique to systematically check whether certain properties hold for different configurations (states) of a system. Provided a transition system $\mathcal{L}$ and formula (or property) $\mathcal{P}$, model checking attempts to verify through refutation that $s \models \mathcal{P}$ for every system state $s \in \mathcal{L}$, resulting in $\mathcal{L} \models \mathcal{P}$. The application of model checking to verify railway signalling systems has a long history within academia and has seen some real applications in industry. One limitation of such model checking is that verification can fail due to over approximation, typically when using techniques such as inductive verification [6]. Here, one solution is to introduce so-called invariants, formal properties satisfied by all states, to suppress false positives [1]. However, automatically generating sufficiently strong invariants to help bound the region of reachable states is complex [2].

In this work, we show it is possible to use machine learning to generate candidate invariants for model checking. Our methodology starts by providing a first formal mapping of state spaces to a reinforcement learning (RL) environment. We then train agents to explore large regions of states spaces while building a dataset of unique state observations. We demonstrate that analysis of state observations gives rise to interesting candidate invariants. Our framework is demonstrated on a set of generated ladder logic programs with known ground truths. Applications to industrial ladder logic programs are then explored in the context of railway interlockings [3]. Overall, this illustrates the viability of RL as a method for invariant proposal, hopefully reducing false positives during verification. While machine learning lacks the completeness guarantees of formal verification, our invariant generation process requires no such promise. As an iterative process, candidate invariants are routinely suggested throughout exploration where the property can then be validated via e.g inductive based model checking.

2 Proposing Invariants using Reinforcement Learning

Interlockings are safety critical systems, functionally a filter between inputs from railway operators to ensure changes to the railway avoid safety conflicts. Ladder logic is a graphical language widely used to implement Programmable Logic Controllers and in particular, industrial interlocking systems. For SAT-based verification, ladder logic diagrams can be abstracted

as propositional formulae [3]. James et al. model ladder logic execution as a labelled transition system (LTS). Within such a setting, we show invariants can be proposed using a two-step approach. First, we devise a mapping that, given an arbitrary ladder logic program, constructs a finite Markov Decision Process (MDP). Reinforcement learning is a machine learning paradigm used to solve sequential decision making problems by modelling the optimal control of some incompletely-known MDP [8]. In RL, the MDP constitutes our training *environment*. Here, RL *agents* aim to maximise state space coverage while generating an observation dataset. Second, we mine this dataset for invariants using statistical measures of variable correlation.

2.1 State Exploration:

Applied to our set of generated ladder logic programs, both single and multi-agent actor-critic algorithms, A2C and A3C [5], resulted in similar levels of state coverage. A3C accelerates training by populating multiple instances of the same environment with independent agents, asynchronously updating a shared behaviour policy with individual experiences. A3C exploration resulted in 100% coverage on programs up to a theoretical size of 2^{30} states, dropping to 91.16% over 2^{40} states, and 50.52% over 2^{50} states. Training sessions where partial coverage was achieved often observed cumulative rewards grow linearly before collapsing to suboptimal performance. This may be due to large network updates shifting parameters into a bad local minimum, failing to recover within the allotted training time. For this reason, we employ trust region models using Proximal Policy Optimisation [7] to better effect, exchanging model update stability for longer training simulations. Applying this approach to an interlocking program, of 2^{6542} theoretical states, reachable state observations increased linearly up to 739372 states over 150 hours of training, without model collapse. While the size of such environments are unknown, metrics such as network entropy and explained variance attribute a quantitative value to predictive uncertainty. Volatility of such measures appear to suggest a high proportion of undiscovered states and thus motivate longer training sessions to populate our dataset.

2.2 Invariant Generation

By omitting state transitions, we can visualise the state space statically as an image. The reachable state matrix, or 'Staterix', shown in Subfigure 1a provides an overview of state valuations, highlighting certain program features. Program variables are read along the x-axis and unique states along the y-axis. Blue cells indicate a value of 0, yellow cells indicate a value of 1. Upon inspection we can explain logical statements present in the program itself and devise properties which appear invariant. For example, the *SingleAspect* invariant proven in [3], namely $\texttt{(TL1G} \vee \texttt{TL1R)} \wedge \neg\texttt{(TL1G} \wedge \texttt{TL1R)}$. While property extraction could be automated through column-wise filtering, we suggest using ϕ correlation coefficients to compute binary relations between pairs of variables. The resulting matrix, shown in Subfigure 1b, allows for more reliable invariant suggestion, where values -1 and 1 indicate an inverse relation or positive relation. For example, the -1 between any green and red lights (traffic or pedestrian) confirms the same *SingleAspect* invariant. Performing row-wise filtering would allow us to build invariants wherever coefficient extrema exist. For example, `TL1G` shares a complete relation with 9 other variables. For each binary pair, determine whether the expression is positive or inverse given the correlation. The 'row-wise' property is then formed by the conjunction of these individual expressions. To compute the correlation between two binary variables, ϕ coefficients [4] require observations from distributions of their valuations. Hence, we expect ϕ to converge over time. Scaling agent rewards according to observations made reducing this uncertainty should motivate agents to pursue new states, prioritising observations which adjust correlation estimates.

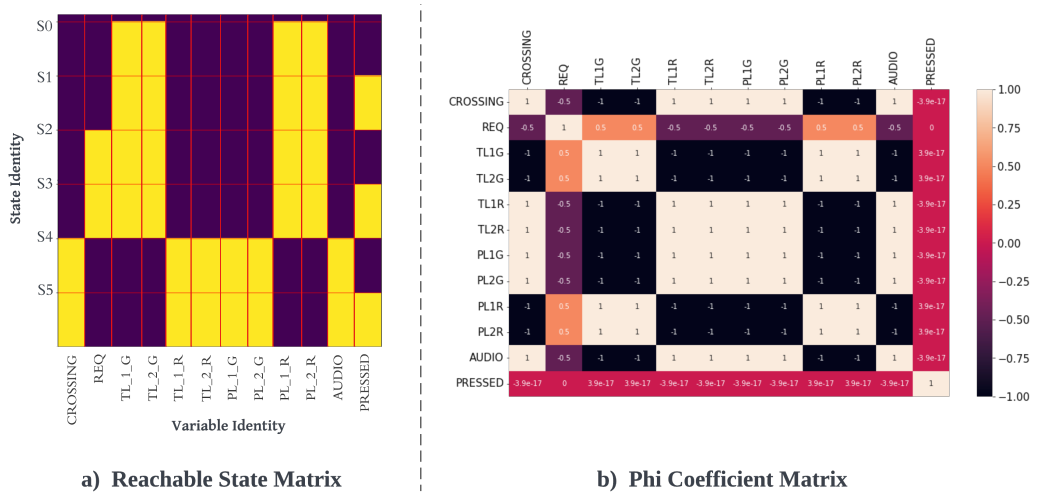

Figure 1: Reachable states and phi correlation coefficients represented as matrices.

3 Conclusion

We have explored the use of reinforcement learning for producing candidate invariants for, but not limited to, ladder logic based state spaces. Our motivation is contextualised by the invariant finding problem for model checking and the reduction of false positives during verification. Overall our results illustrate the viability of machine learning as a means of suggesting invariants for model checking. In future we aim to explore pattern mining over all agent observations to identify invariants across state sequences. Temporal clustering of unique sequences may also allow mining for persistent state sequences, possibly indicative of event-bound invariants.

References

[1] M. Awedh and F. Somenzi. Automatic invariant strengthening to prove properties in bounded model checking. In *ACM/IEEE Design Automation Conference*, 2006.

[2] G. Cabodi, S. Nocco, and S. Quer. Strengthening model checking techniques with inductive invariants. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2009.

[3] 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.

[4] B. W. Matthews. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure*, 405, 1975.

[5] 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.

[6] H. Post, C. Sinz, A. Kaiser, and T. Gorges. Reducing false positives by combining abstract interpretation and bounded model checking. In *IEEE/ACM International Conference on Automated Software Engineering*, 2008.

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

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